Package: nmslibR (via r-universe)

October 18, 2026

Type Package

Title Non Metric Space (Approximate) Library

Version 1.0.7

Date 2023-02-01

BugReports https://github.com/mlampros/nmslibR/issues

URL https://github.com/mlampros/nmslibR

Description A Non-Metric Space Library ('NMSLIB' <https://github.com/nmslib/nmslib>) wrapper, which according to the authors ``is an efficient cross-platform similarity search library and a toolkit for evaluation of similarity search methods. The goal of the 'NMSLIB' <https://github.com/nmslib/nmslib> Library is to create an effective and comprehensive toolkit for searching in generic non-metric spaces. Being comprehensive is important, because no single method is likely to be sufficient in all cases. Also note that exact solutions are hardly efficient in high dimensions and/or non-metric spaces. Hence, the main focus is on approximate methods". The wrapper also includes Approximate Kernel k-Nearest-Neighbor functions based on the 'NMSLIB' <https://github.com/nmslib/nmslib> 'Python' Library.

License Apache License (>= 2.0)

SystemRequirements python3-dev: apt-get install -y python3-dev (deb), python3-pip: apt-get install -y python3-pip (deb), numpy: pip3 install numpy (deb), scipy: pip3 install scipy (deb), nmslib: pip3 install --no-binary :all: nmslib (deb)

Encoding UTF-8

Depends R(>= 3.2.3)

Imports Rcpp (>= 0.12.7), reticulate, R6, Matrix, KernelKnn, utils, lifecycle

LinkingTo Rcpp, RcppArmadillo (>= 0.8.0)

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.2.3

Config/reticulate list(packages = list(list(package = ``nmslib'', pip = TRUE), list(package = ``scipy'', pip = TRUE)))

NeedsCompilation yes

Author Lampros Mouselimis [aut, cre] (<https://orcid.org/0000-0002-8024-1546>), B. Naidan [cph] (Author of the Non-Metric Space Library (NMSLIB)), L. Boytsov [cph] (Author of the Non-Metric Space Library (NMSLIB)), Yu. Malkov [cph] (Author of the Non-Metric Space Library (NMSLIB)), B. Frederickson [cph] (Author of the Non-Metric Space Library (NMSLIB)), D. Novak [cph] (Author of the Non-Metric Space Library (NMSLIB))

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

Config/pak/sysreqs python3

Repository https://cran.r-universe.dev

Date/Publication 2023-02-01 18:20:02 UTC

RemoteUrl https://github.com/cran/nmslibR

RemoteRef HEAD

RemoteSha ebdc792b04137e6f205cb607bec8ddda2e4a4c7f

Contents

KernelKnn_nmslib 2
KernelKnnCV_nmslib 4
mat_2scipy_sparse 6
NMSlib 7
TO_scipy_sparse 10

Index **13**

`KernelKnn_nmslib` *Approximate Kernel k nearest neighbors using the nmslib library*

Description

Approximate Kernel k nearest neighbors using the nmslib library

Usage

```
KernelKnn_nmslib(
  data,
  y,
  TEST_data = NULL,
  k = 5,
  h = 1,
  weights_function = NULL,
  Levels = NULL,
  Index_Params = NULL,
  Time_Params = NULL,
  space = "l1",
  space_params = NULL,
  method = "hnsw",
  data_type = "DENSE_VECTOR",
  dtype = "FLOAT",
  index_filepath = NULL,
  print_progress = FALSE,
  num_threads = 1
)
```

Arguments

data	either a matrix or a scipy sparse matrix
y	a numeric vector specifying the response variable (in classification the labels must be numeric from 1:Inf). The length of *y* must equal the rows of the *data* parameter
TEST_data	a test dataset (in case of a matrix the *TEST_data* should have equal number of columns with the *data*). It is assumed that the *TEST_data* is an unlabeled dataset
k	an integer. The number of neighbours to return
h	the bandwidth (applicable if the weights_function is not NULL, defaults to 1.0)
weights_function	there are various ways of specifying the kernel function. See the details section.
Levels	a numeric vector. In case of classification the unique levels of the response variable are necessary
Index_Params	a list of (optional) parameters to use in indexing (when creating the index)
Time_Params	a list of parameters to use in querying. Setting *Time_Params* to NULL will reset
space	a character string (optional). The metric space to create for this index. Page 31 of the manual (see *references*) explains all available inputs
space_params	a list of (optional) parameters for configuring the space. See the *references* manual for more details.
method	a character string specifying the index method to use
data_type	a character string. One of 'DENSE_UINT8_VECTOR', 'DENSE_VECTOR', 'OBJECT_AS_STRING' or 'SPARSE_VECTOR'

dtype	a character string. Either 'FLOAT' or 'INT'
index_filepath	a character string specifying the path to a file, where an existing index is saved
print_progress	a boolean (either TRUE or FALSE). Whether or not to display progress bar
num_threads	an integer. The number of threads to use

Details

There are three possible ways to specify the *weights function*, 1st option : if the weights_function is NULL then a simple k-nearest-neighbor is performed. 2nd option : the weights_function is one of 'uniform', 'triangular', 'epanechnikov', 'biweight', 'triweight', 'tricube', 'gaussian', 'cosine', 'logistic', 'gaussianSimple', 'silverman', 'inverse', 'exponential'. The 2nd option can be extended by combining kernels from the existing ones (adding or multiplying). For instance, I can multiply the tricube with the gaussian kernel by giving 'tricube_gaussian_MULT' or I can add the previously mentioned kernels by giving 'tricube_gaussian_ADD'. 3rd option : a user defined kernel function

Examples

```
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("nmslib")) {

      library(nmslibR)

      x = matrix(runif(1000), nrow = 100, ncol = 10)

      y = runif(100)

      out = KernelKnn_nmslib(data = x, y = y, k = 5)
    }
  }
}, silent=TRUE)
```

KernelKnnCV_nmslib *Approximate Kernel k nearest neighbors (cross-validated) using the nmslib library*

Description

Approximate Kernel k nearest neighbors (cross-validated) using the nmslib library

Usage

```
KernelKnnCV_nmslib(
  data,
  y,
  k = 5,
  folds = 5,
  h = 1,
```

```
  weights_function = NULL,
  Levels = NULL,
  Index_Params = NULL,
  Time_Params = NULL,
  space = "l1",
  space_params = NULL,
  method = "hnsw",
  data_type = "DENSE_VECTOR",
  dtype = "FLOAT",
  index_filepath = NULL,
  print_progress = FALSE,
  num_threads = 1,
  seed_num = 1
)
```

Arguments

data	a numeric matrix
y	a numeric vector specifying the response variable (in classification the labels must be numeric from 1:Inf). The length of *y* must equal the rows of the *data* parameter
k	an integer. The number of neighbours to return
folds	the number of cross validation folds (must be greater than 1)
h	the bandwidth (applicable if the weights_function is not NULL, defaults to 1.0)
weights_function	there are various ways of specifying the kernel function. See the details section.
Levels	a numeric vector. In case of classification the unique levels of the response variable are necessary
Index_Params	a list of (optional) parameters to use in indexing (when creating the index)
Time_Params	a list of parameters to use in querying. Setting *Time_Params* to NULL will reset
space	a character string (optional). The metric space to create for this index. Page 31 of the manual (see *references*) explains all available inputs
space_params	a list of (optional) parameters for configuring the space. See the *references* manual for more details.
method	a character string specifying the index method to use
data_type	a character string. One of 'DENSE_UINT8_VECTOR', 'DENSE_VECTOR', 'OBJECT_AS_STRING' or 'SPARSE_VECTOR'
dtype	a character string. Either 'FLOAT' or 'INT'
index_filepath	a character string specifying the path to a file, where an existing index is saved
print_progress	a boolean (either TRUE or FALSE). Whether or not to display progress bar
num_threads	an integer. The number of threads to use
seed_num	a numeric value specifying the seed of the random number generator

Details

There are three possible ways to specify the *weights function*, 1st option : if the weights_function is NULL then a simple k-nearest-neighbor is performed. 2nd option : the weights_function is one of 'uniform', 'triangular', 'epanechnikov', 'biweight', 'triweight', 'tricube', 'gaussian', 'cosine', 'logistic', 'gaussianSimple', 'silverman', 'inverse', 'exponential'. The 2nd option can be extended by combining kernels from the existing ones (adding or multiplying). For instance, I can multiply the tricube with the gaussian kernel by giving 'tricube_gaussian_MULT' or I can add the previously mentioned kernels by giving 'tricube_gaussian_ADD'. 3rd option : a user defined kernel function

Examples

```
## Not run:

x = matrix(runif(1000), nrow = 100, ncol = 10)

y = runif(100)

out = KernelKnnCV_nmslib(x, y, k = 5, folds = 5)


## End(Not run)
```

`mat_2scipy_sparse` *conversion of an R matrix to a scipy sparse matrix*

Description

conversion of an R matrix to a scipy sparse matrix

Usage

```
mat_2scipy_sparse(x, format = "sparse_row_matrix")
```

Arguments

`x`	a data matrix
`format`	a character string. Either *"sparse_row_matrix"* or *"sparse_column_matrix"*

Details

This function allows the user to convert an R matrix to a scipy sparse matrix. This is useful because the *nmslibR* package accepts only *python* sparse matrices as input.

References

https://docs.scipy.org/doc/scipy/reference/sparse.html

Examples

```
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("scipy")) {

      library(nmslibR)

      set.seed(1)

      x = matrix(runif(1000), nrow = 100, ncol = 10)

      res = mat_2scipy_sparse(x)

      print(dim(x))

      print(res$shape)
    }
  }
}, silent=TRUE)
```

NMSlib *Non metric space library*

Description

Non metric space library

Non metric space library

Usage

```
# init <- NMSlib$new(input_data, Index_Params = NULL, Time_Params = NULL,
#                            space='l1', space_params = NULL, method = 'hnsw',
#                            data_type = 'DENSE_VECTOR', dtype = 'FLOAT',
#                            index_filepath = NULL, load_data = FALSE,
#                            print_progress = FALSE)
```

Details

input_data parameter : In case of numeric data the *input_data* parameter should be either an R matrix object or a scipy sparse matrix. Additionally, the *input_data* parameter can be a list including more than one matrices / sparse-matrices having the same number of columns (this is ideal for instance if the user wants to include both a train and a test dataset in the created index)

the *Knn_Query* function finds the approximate K nearest neighbours of a vector in the index

the *knn_Query_Batch* Performs multiple queries on the index, distributing the work over a thread pool

the *save_Index* function saves the index to disk

If the *index_filepath* parameter is not NULL then an existing index will be loaded

Incrementally updating an already saved (and loaded) index is *not* possible (see: https://github.com/nmslib/nmslib/issues/73)

Methods

```
NMSlib$new(input_data, Index_Params = NULL, Time_Params = NULL, space='l1', space_params = NULL, method =
```

———

```
Knn_Query(query_data_row, k = 5)
```

———

```
knn_Query_Batch(query_data, k = 5, num_threads = 1)
```

———

```
save_Index(filename, save_data = FALSE)
```

Methods

Public methods:

- NMSlib$new()
- NMSlib$Knn_Query()
- NMSlib$knn_Query_Batch()
- NMSlib$save_Index()
- NMSlib$clone()

Method new():

Usage:

```
NMSlib$new(
  input_data,
  Index_Params = NULL,
  Time_Params = NULL,
  space = "l1",
  space_params = NULL,
  method = "hnsw",
  data_type = "DENSE_VECTOR",
  dtype = "FLOAT",
  index_filepath = NULL,
  load_data = FALSE,
  print_progress = FALSE
)
```

Arguments:

`input_data` the input data. See *details* for more information

`Index_Params` a list of (optional) parameters to use in indexing (when creating the index)

`Time_Params` a list of parameters to use in querying. Setting *Time_Params* to NULL will reset

`space` a character string (optional). The metric space to create for this index. Page 31 of the manual (see *references*) explains all available inputs

`space_params` a list of (optional) parameters for configuring the space. See the *references* manual for more details.

`method` a character string specifying the index method to use

`data_type` a character string. One of 'DENSE_UINT8_VECTOR', 'DENSE_VECTOR', 'OBJECT_AS_STRING' or 'SPARSE_VECTOR'

`dtype` a character string. Either 'FLOAT' or 'INT'

`index_filepath` a character string specifying the path to a file, where an existing index is saved

`load_data` a boolean. If TRUE then besides the index also the saved data will be loaded. This parameter is used when the *index_filepath* parameter is not NULL (see the web links in the *references* section for more details). The user might also have to specify the *skip_optimized_index* parameter of the *Index_Params* in the "init" method

`print_progress` a boolean (either TRUE or FALSE). Whether or not to display progress bar

Method `Knn_Query()`**:**

Usage:

```
NMSlib$Knn_Query(query_data_row, k = 5, include_query_data_row_index = FALSE)
```

Arguments:

`query_data_row` a vector to query for

`k` an integer. The number of neighbours to return

`include_query_data_row_index` a boolean. If TRUE then the index of the query data row will be returned as well. It currently defaults to FALSE which means the first matched index is excluded from the results (this parameter will be removed in version 1.1.0 and the output behavior of the function will be changed too - see the deprecation warning)

Method `knn_Query_Batch()`**:**

Usage:

```
NMSlib$knn_Query_Batch(query_data, k = 5, num_threads = 1)
```

Arguments:

`query_data` the query_data parameter should be of the same type with the *input_data* parameter. Queries to query for

`k` an integer. The number of neighbours to return

`num_threads` an integer. The number of threads to use

Method `save_Index()`**:**

Usage:

```
NMSlib$save_Index(filename, save_data = FALSE)
```

Arguments:

`filename` a character string specifying the path. The filename to save (in case of the *save_Index* method) or the filename to load (in case of the *load_Index* method)

`save_data` a boolean. If TRUE then besides the index also the data will be saved (see the web links in the *references* section for more details)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NMSlib$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://github.com/nmslib/nmslib/blob/master/manual/latex/manual.pdf

https://github.com/nmslib/nmslib/blob/master/python_bindings/notebooks/search_vector_dense_optim.ipynb

https://github.com/nmslib/nmslib/blob/master/python_bindings/notebooks/search_vector_dense_nonoptim.ipynb

https://github.com/nmslib/nmslib/issues/356

https://github.com/nmslib/nmslib/blob/master/manual/methods.md

https://github.com/nmslib/nmslib/blob/master/manual/spaces.md

Examples

```
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("nmslib")) {

      library(nmslibR)

      set.seed(1)
      x = matrix(runif(1000), nrow = 100, ncol = 10)

      init_nms = NMSlib$new(input_data = x)


      # returns a 1-dimensional vector (index, distance)
      #------------------------------------------------

      init_nms$Knn_Query(query_data_row = x[1, ], k = 5)


      # returns knn's for all data
      #---------------------------

      all_dat = init_nms$knn_Query_Batch(x, k = 5, num_threads = 1)
    }
  }
}, silent=TRUE)
```

TO_scipy_sparse *conversion of an R sparse matrix to a scipy sparse matrix*

Description

conversion of an R sparse matrix to a scipy sparse matrix

Usage

```
TO_scipy_sparse(R_sparse_matrix)
```

Arguments

R_sparse_matrix
: an R sparse matrix. Acceptable input objects are either a *dgCMatrix* or a *dgRMatrix*.

Details

This function allows the user to convert either an R *dgCMatrix* or a *dgRMatrix* to a scipy sparse matrix (*scipy.sparse.csc_matrix* or *scipy.sparse.csr_matrix*). This is useful because the *nmslibR* package accepts besides an R dense matrix also python sparse matrices as input.

The *dgCMatrix* class is a class of sparse numeric matrices in the compressed, sparse, *column-oriented format*. The *dgRMatrix* class is a class of sparse numeric matrices in the compressed, sparse, *column-oriented format*.

References

https://stat.ethz.ch/R-manual/R-devel/library/Matrix/html/dgCMatrix-class.html, https://stat.ethz.ch/R-manual/R-devel/library/Matrix/html/dgRMatrix-class.html, https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse

Examples

```
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("scipy")) {

      if (Sys.info()["sysname"] != 'Darwin') {

        library(nmslibR)


        # 'dgCMatrix' sparse matrix
        #--------------------------

        data = c(1, 0, 2, 0, 0, 3, 4, 5, 6)

        dgcM = Matrix::Matrix(data = data, nrow = 3,

                              ncol = 3, byrow = TRUE,

                              sparse = TRUE)

        print(dim(dgcM))

        res = TO_scipy_sparse(dgcM)

        print(res$shape)


        # 'dgRMatrix' sparse matrix
        #--------------------------
```

```
          dgrM = as(dgcM, "RsparseMatrix")

          print(dim(dgrM))

          res_dgr = TO_scipy_sparse(dgrM)

          print(res_dgr$shape)
        }
      }
    }
  }, silent=TRUE)
```

Index

KernelKnn_nmslib, 2
KernelKnnCV_nmslib, 4

mat_2scipy_sparse, 6

NMSlib, 7

TO_scipy_sparse, 10